

プログラム習得＝構成技法と弁明の習得

プログラミング教育における効果的な問題解決技術は、入門コンピュータプログラミングのための履修課程変更を必要としている。

エリット・ソロウェイ

自発性と目標

人体認知科学が成長して専門技術の本質となっていることが革命の火に油を注ぎ、今日の科学教科課程と数学教科課程の根本的な改訂を引き起こしている。過去に教えられてきたことは実際に専門家が知っていることではない。例えば、幾何学の生徒は教師が黒板に一行ずつ書いた時、その証明段階を実際に理解する。しかしながら宿題で証明を解かせると、生徒はどこから始めればよいかわからないことがよくある。なぜだろう？数学者はきちんと線形方法で証明を解かない。もっと適切に言えば、証明解法は非線形的な解析過程である。残念ながら生徒は、非線形的である証明解法の本質を告げられていない。彼らは教師の証明解法を一行ずつ理解し、当然のように、彼らは同じようにできなければならないと思っている。今日、幾何学証明のような教育的論題は、証明解法を導く発見的学習のような明快な教育を含むよう改訂されている。

この論説は、履修課程の懸命な見直しとつながり、入門プログラミング課程で教えられるべき概念に焦点をおいている。入門プログラミング課程で用いられる教科書は通常、言語の構文と意味構成に焦点をおく。しかしながら、初心者プログラマーとの新たな研究は、言語構造がプログラムを学ぶ初心者にとって大きな障害ではないことを暗示している。もっと適切に言えば、初心者が持つ実際の問題は組み立てとプログラム部品調整、つまり“部品をまとめること”にある。専門的なプログラマーは単なる構文や言語構造の意味だけでなく、もっと多くのことを知っている。彼らは決まりきった問題解決の蔵書を増やしていき、調整と組み立ての方策も同様に蓄えていく。生徒はプログラムを使いこなすために、それらの蔵書や方策について明示的に教えられなければならない。

学校でプログラミングを教えることは特に目新しい話題ではない。だがプログラミングは単に職業技術であり、プログラミング教育は一般の履修課程に含まれるべきではないと議論される一方、他方ではプログラミングは人が効果的な問題解決技術を学ぶことができる教科であるとして議論されている。現在、後者のための経験的な援助が全くない。しかしながら、問題解決におけるプログラミング効果の欠如は、プログラミングの基礎をなすその重要な考えを生徒が教わっていないという事実のせいかも

しれない。パスカルでセミコロンをどこに置くかを学ぶことが、いったい問題解決能力の増進につながるのだろうか？

構文とプログラミング言語構造の意味を教える教育への注目は、プログラミング過程の出力プログラムに重点を置いてしまう。しかしながら、プログラムには2つの支持がある。

- コンピュータ：プログラム教育はコンピュータを、どのように問題を解決するかを命令する装置へと変える。
- 読み手：プログラマーはプログラムが与えられた問題を解決する弁明を必要とする。

この意見は、結果的にプログラム学習が装置と弁明の構成方法を学ぶことになる。装置構成方法と弁明構成方法の必要性はプログラミング領域を越える。日常の生活で、人は問題解決のための技法と弁明を発達させている。例えば、空港から会社への道を来客に教えることは技法であり、なぜその経路が効果的な選択であるかの理由を与えることは弁明である。入門プログラミング課程において、生徒はプログラミングについて本当に知る必要があることを教えられるべきだ。そうすることで、生徒はプログラミング領域を越える、効果的な問題解決方を学ぶことができるであろう。

基本的前提

履修課程提案の基礎をなす2つの重要な前提は以下の通りである。

暗黙の認識。専門家は彼らが問題解決やプログラミングなどに利用する知識や方策を必ずしも認識しているわけではない。しばしば、専門家はインスピレーションの源として、直感や本能的な勘といったものに頼る。しかしながら、科学者の仕事は専門家が用いる他の暗黙の認識を探り出すために、どこが暗黙なのかを明確にすることである。この科学者の仕事は、数学や物理学などといった様々な幅広い領域で、実際に実施されてきた。良いプログラミングと良い問題解決に結びつく認識の新しい型としてだけでなく、むしろすでにあったことを簡単に明快にするとして、読者は次の節を読めば、そこで描写されている概念の大部分を直ちに理解できるであろう。

フォーフィアン仮説。暗黙の概念を探り出し、明快に教えるべきだとするこの考えはある程度フォーフィアン仮説に留まっている。言語表記において、ベンジャミン・フォーフは“思考設計言語”を提案した。これは、そのための単語を持っているか、いないかを考えるだけでいい。フォーフの推測意見における厳密直訳な解釈は強烈すぎるように思われる。しかしながら、“思考援助言語”は確かに多大な魅力を持つと主張

する少数派がいる。生徒が学ぶべきことを明確に教えられない時、彼らはどうやって概念を学ぶことができるのか？先ほど述べた幾何学的証明例は、この問題の格好の例である。

従って、専門用語事項のまとまりは、全般的に見て、専門家の頭にある暗黙の認識として残っているが、明快に教えられ、説明される必要がある。

価値ある専門用語集

専門家が用いる多くの知識と方策は、入門プログラミング課程の生徒に明示的に教えられる必要がある。概念には大まかに 2 つの区分がある。知識を使うための知識と方策である。次の一覧は完全ではない。基本的な主題を説明しようとしただけである。

目標と plans：物事の真意

問題解析とプログラム構成に生じる障害は何であろうか？目標と plans—決まりきった、月並みの解決方法—は同様な働きにおいて、2 つの重要な要素である。簡単な問題分解のくだけた立ち稽古は、目標と plans の本質である直感的な理解を与える。もっと正式な専門用語はこの節の終わりに紹介する。

それでは、続く問題に有力な要素とは何であろうか？

標準問題： 整数を読み込み、それらの平均値を出力するプログラムを書きなさい。
99999 が入力されたら読み込みを停止しなさい。

平均は選択された入力数字の合計を必要とする。そしてこの合計はたされた数字の個数で割られなければならない。これら 2 つの要求が問題の目標である。この問題解決プログラミング開発には、少なくともさらに 2 つの付加目標が出されなければならない。：数字は入力されなければならないし、平均値は出力されなければならない。専門プログラマーは、これら 4 つの目標が多くのプログラムによく現れるので、解析のために標準的な技法を開発してきた。例えば、プログラマーは連続する数字を読み込んで合計し、見張り値が入力されると停止させるために、テンプレート構造を用いる。

running total を初期化する。

ユーザーに値を求める。

もし入力が見張り値ではなかったら

新しい値を running total へ加える。

入力に戻って繰り返す。

このテンプレートは言語に従属していない。専門家はこのテンプレートを Pascal, C, PL/1 などて例示することができる。我々の専門用語では、そのような月並みな解決方法を plans という。プログラマーはまた読み込みと入力値数計算、さらに見張り値入力時停止のために、“繰り返し plan”を用いる。

```
counter を初期化する。  
ユーザーに値を要求する。  
もし入力が見張り値ではなかったら  
    counter 値を 1 増やす。  
    入力に戻って繰り返す。
```

特に注意すべき点は、“和算目標解析 plan”と“カウント目標解析 plan”を結合しなければならないことである。なぜなら、データは 1 度だけ通されなければならないので。

```
running total を初期化する。  
counter を初期化する。  
ユーザーに値を要求する。  
もし入力が見張り値ではなかったら  
    新しい値を running total へ加える。  
    counter 値を 1 増やす。  
    入力に戻って繰り返す。
```

和とカウントを一度だけ計算し、平均値を求めるために割られなければならない。専門プログラマーは、割り算で 0 をとると実行時エラーが生じるので、その割り算を保護しなければならないことを知っている。

```
もしカウントが 0 以上なら  
    そのときは計算する。  
    そうでなかったら問題をユーザーに報告する。
```

この“保護 plan”は先ほどの“和算とカウント plan”の直後にくる。このような plan 合成形式をアバットメント(接合)と呼ぶ。

たとえ先ほどの議論が直接的であろうとも、目標能力である plan 言語は次の 2 つの例に見受けられる。

例 1： 以下の問題を考察せよ。

全ての入力が“T”であるとき“T”を出力し、入力の中にひとつでも“F”があれば“F”を出力するプログラムを書け。“#”が入力されたら読み込みを停止せよ。

見かけ上、この問題は先ほどの標準問題とは異なる。しかしながらこの問題は標準問題とほとんど同形の plan 構造である。見張り値は入力読み込みを停止させるのに使う。このプログラムは平均を求める問題のような累算合計の代わりに、フラッグ(標識)を設定し、入力値により再設定される必要がある。なぜなら文字を指定しない入力もありえるので、出力保護を行わなければならないからである。プログラミング技術—あるいは通常の問題解決—において重要なことは、有用な抽象メソッドを開発することである。もし解決すべき問題が新たなものかそうでないかを生徒が見分けられれば、経験を再利用することができる。専門技術の品質証明は、古くからの問題専用語で新しい問題を考察するための能力である。だから解決方策は古い状態から今の状況に移し返られる。目標と plan 言語は、まさに問題と解決方法を述べるための言語を与える。

例 2：毎日の降水量の数値を読み込み、日割り降水量の平均値を表示させなさい。降水量の数値が 0 以下だったら、ユーザーに新たな降水量を促しなさい。

BUGGY VERSION	CORRECT VERSION
BEGIN	BEGIN
sum := 0;	sum := 0;
count := 0;	count := 0;
writeln('please input a rainfall');	writeln('please input a rainfall');
read(rainfall);	read(rainfall);
while rainfall < 0 do	while rainfall <> 99999 do
begin	begin
writeln('rainfall cannot be < 0;	while rainfall < 0 do
input again');	begin
read(rainfall);	writeln('rainfall cannot be < 0;
end;	input again');
while rainfall <> 99999 do	read('rainfall');
begin	end;
sum := sum + rainfall;	sum := sum + rainfall;
count := count + 1;	count := count + 1;
writeln('please input a rainfall');	writeln('please input a rainfall');
read(rainfall);	read(rainfall);
read(rainfall);	read(rainfall);

end;	end;
etc.	etc.
END.	END.

上図は先ほどの平均値プログラムよりもっと複雑なバージョンを、生徒が作った虫つきプログラムである。このバグのよい表現は何であろうか？なぜ生徒はこのバグを作ってしまったのだろうか？目標と plan 言語の有用性が再び判明する。生徒は“和算カウンタ plan”に“妥当データ入力 plan”を結合していない。“妥当データ入力 plan”とはろ過性の繰り返し plan であり、入力データの妥当性を調べる。実際、これらの plan 結合はかなりの技術を要する。文字通り何千もの生徒分析は、まさにどこを結合すべきかといった状況で、生徒は重大な困難を抱えているというプログラム提案を生み出す。もっと多くの目標と plan の使用例は[11]と[37]で取り上げている。

目標と plan 言語が、コンピュータプログラム開発と解析に有用であることは単なる一致ではない。経験情報はプログラミング、チェス、碁、電子回路、数学などの分野で専門家が問題解決において用いる目標と plans の理論要求を助ける。この分野における名高い実験は、チェスプレイヤーのチェイスとサイモンによって行われた。

熟達者と初心者のチェスプレイヤーは重要なチェス盤を見せられ、いったん盤を隠した後、そのこまを思い出すよう告げられる。熟達者は初心者より多くのこまを思い出す。次に、熟達者と初心者はチェスのこまがランダムに置かれた盤を見せられると、どうなるであろうか？熟達者の成果は初心者と同じになった。

熟達者が重要なチェス盤を初心者より思い出すことができたのは、重要な構成単位の代わりとなる専門用語のまとまりに変換することができたからだ。チェイスとサイモンは論じた。反対に、当然ながら初心者は熟達者が持つ経験がほとんどないので、特定なまとまりにする区分能力がない。同様な経験はプログラミングにおいても同じ結果を引き起こす。この種の経験は、専門家が特定な区分にするときの理論を手助けする。これらのまとまりとは、プログラミングにおける plan である。最終的に、このまとまり(plan)の概念は、技術的な問題分野だけに有用であるわけではない。認知心理学者は長い間、概念的構成単位として先験的図式の存在を断定してきた。コンピュータプログラムの読み書きにおける plan は、物語の読み書きにおいても同じ働きをする。このように、目標と plan 概念の基礎は心理学研究にある。

目標と plan はより具体的な達成を与える。例えば、“読み込んで累算、99999 入力時に停止”という plan は「見張り制御合計繰り返し実行 PLAN」と呼ばれる。同様に、

“読み込んだ個数を数え、99999 入力時に停止”という plan は、「見張り制御カウンター繰り返し PLAN」と呼ぶ。平均値計算における割り算保護は、「スキップ保護 PLAN」と呼ぶ。その目的は plan の目標観念を名称で伝達するためである。例え plan 名がわずらわしいと控えめに言っても、問題解決で何度も用いられる構造に、明快なラベルを与える。ラベル割り当ては plan の確立を手助けする。実際に、全ての plan ライブラリーは入門プログラミング教育のために作られてきた。生徒はこのライブラリーの plan を使うために、明快な教育を受け、自身で plan を構成できるようになる。目標と plan 言語は、入門プログラミング教育における履修改訂提案の核心である。

技法と弁明：プログラミング生成過程

もし入門プログラミング課程で職業技術以上のことを生徒に教えるとしたら、プログラミングの基礎的抽象概念を明快にしなければならない。プログラミングは他の問題解決作業と同様であることを生徒に教えるべきだ。プログラミング過程の出力としてプログラムを注目することで、生徒は自然に「“コンピュータ科学 100”で学んだことはプログラム生成だけに関することだ」と考える。

知識変換を手助けする。“コンピュータ科学 100”を他の問題解決知識に変換する手助けをするために、プログラミングが設計訓練とプログラミング過程の出力であるだけでなく、本質的にむしろ、要求機能を行う人工物であるということを生徒は明快に教えられなければならない。この人工物は技法の概念を導く。トランジスタ、ボルト、ギア、取扱説明書で例示しようとも、技法とは要求動作結果を生み出す装置の動作を説明する。ソフトウェアで例示される人工物は、もっぱらの変更にとっても適応する。プログラマーは通常、1 回限りの人工物を作るような無駄なことはしない。むしろ、次のプログラマーがどのように、なぜ人工物は効果的な変更を行えるような方法で設計されたかの手がかりを、プログラマーは与える必要がある。この手がかり情報は弁明である。プログラミング過程の成果がまさに技法と弁明である。生徒が入門プログラミング課程で学ぶことは、技法と弁明を構築するための知識と技術である。

図 2 の右半分は平均値問題のパスカルプログラムである。この問題は平均値計算方法に関するコンピュータ構成技法である。

図 2 のプログラムはどのように要求問題を解決しているか？コンピュータ(あるいはコンピュータの役割を引き受ける人間)に実行されるとき、プログラムはユーザーから数字を受け取り、total と呼ばれる累算器に数字を足していくなどする。

図 2 の記述は技法表現の一部分だけではない。技法の実行において、コンピュータは紙に書かれた静的プログラムを、オーバータイムがある動的実体に変える。この動的表現間の因果関係概念はとても重要で、プログラムの働き方を表現するのに用いら

れるべきである。例えば、累算とカウンターの更新は、次の入力を読み込まれる前に行われる。見張り値が入力されると、カウンターが0より大きいかを調べる。

図2の左半分は、なぜプログラムが要求された答えを計算するかという弁明である。英語表現の問題目標は同一視され、決まりきったメソッド(plan)はそれらの目標解決に用いられる。

なぜプログラムは入力数値平均を効果的に計算するのだろうか？「見張り制御合計繰り返し実行 PLAN」はここで使われる。なぜならこの問題は、特定値入力に従った繰り返し停止を要求するので、それと累算はプログラムのユーザーからの入力数値和算目標を必要とするから。さらに、平均値計算のために、「見張り制御合計繰り返し実行 PLAN」と「見張り制御カウンター繰り返し PLAN」の結合は、入力値情報を見失わないようにする。なぜなら入力データは1回だけ通過するので。

弁明では、“なぜなら”“するために”といった語句が用いられる。これらの語句は、問題目標とプログラム例示 plan 間関係説明情報の存在を知らせる。プログラム記述が技法全体を伝えないように、図2の技法と弁明の概念は重要な情報をいくつか含んでいない。例えば、どのようにプログラマーは、「カウンター繰り返し PLAN」と「見張り制御合計繰り返し PLAN」の結合が訂正されることを知るのだろうか？あるいは、なぜプログラマーは入力を配列に読み込む、「見張り制御配列入力繰り返し PLAN」を用いた処理方法を選択しないのであろうか？この情報はもっと完全に説明されなければならない。

技法と弁明についての話し方を理解することは、それでも初期の段階である。少なくとも技法についての会話には、構成言語、データ制御フロー、静的属性、動的属性などといった用語がすでに存在する。類似役割、目標論議、plan、そしてそれらの関係は、弁明議論のための具体的な言葉を与える。

設計方策：構成技法と弁明

漸次改良

漸次改良は問題考察のための、命令メソッドを与える構想技術である。ほとんど全てのプログラミング教科書は、この技術の使い方を、生徒に教えることを試みる。通常生徒は問題を分割するよう告げられる。作者がどのようにこの指示を実行するか、生徒は実例を挙げられる。ほとんどの生徒はこの議論をとてもよく理解できる。そして行程変化も理解することができる。しかしながら、他の問題で漸次改良を構成しな

さいと言うと、ほとんどの生徒がどこから始めたらいいか分からなくなる。漸次改良を生徒が試みると、実際にどのようにすればいいのかという疑問に直面し、困難が生じる。なぜこの問題は n 個に分解されるのだろうか？なぜ m 個だと駄目なのか？それになぜこんな特定の手順を使わなければならないのか？なぜ他の手順だと駄目なのか？教科書は大抵、これらの疑問に答えを与えない。生徒は教科書の例題に基づく漸次改良方策をする気が失せる。

明快ではないが直感的にわかる発見的教授法を、漸次改良教育に加えなければならない。

過去に解いた問題を土台として、既知の決まりきった解法で、問題を分解しなさい。

漸次改良教育で、我々が表現しようとする概念は以下の通りである。

あなたはすでに、多くの決まりきった解法を備えていると仮定する。その多くの解法から問題を調べ、新たな問題の解決に使えないかどうか知ろうとする。決まりきった解法を使えるようにするために、新たな問題を分解する。

言い換えれば、生徒は漸次改良方策を行うために、初期的な問題分解能力をすでに備えていなければならない。

決まりきった解法はプログラミング PLAN である。すなわち達成目標のための決まりきったメソッドである。漸次改良手順方策の目標は、与えられた問題を開発済みのプログラミング PLAN で認識することである。しかし、問題に関する PLAN を生徒が分からなかったらどうなるだろうか？例えば、生徒がプログラミングを学び始めたときだったら、どうなるだろうか？答えは簡単だ。生徒は簡単に漸次改良に携われない。生徒はかなり混乱しながら勉強し、問題に不適当な分割をしがちである。実際ソフトウェア設計 1 年生の授業では、まさにこの混乱作用が見受けられる。

弁明の大部分が、技法の基礎をなす目標と計画からなっていることを思い出してほしい。漸次改良過程は実際、なぜ結果として生じる技法が問題解法に効果的なのかを明らかにする、目標と PLAN の分解を行っていることに気付いてほしい。目標と PLAN の分解は、完成に目標達成技術(plan)を必要とする目標と関連する。目標と PLAN の分解の一部として、なぜある目標(あるいは plan)が他よりも優先して選ばれるのかという理由もまた、明快にしなければならない。

合成メソッド plan

漸次改良は与えられた問題を、目標と計画を分解して開発するためのマクロ方策を述べる。しかしながら、どのようなミクロレベルで plan を組み合わせるべきかという

規則も確認される。例えば、漸次改良による平均値問題解法では、出力目標と累算目標は、計算数値の表示と平均値計算を必要とする。なぜこの2つの目標を実現する plan は合成されなければならないのだろうか？なぜなら累算 PLAN は出力 PLAN が行われる前に完全に終了されてしまうから。2つの plan は単に隣接しているだけだ。平均値計算 plan は平均値出力 plan への入力値を出力する。

一般に、合成 plan のために4つの方策が確認されている。

- 接合：平均値問題で例示されたように、2つの plan は順番に前後逆に合成される。
- 入れ子：ある plan が他の plan に完全に包囲される。例えば、図2の平均値問題で、平均値出力目標である出力 plan は、スキップ保護 plan 内に入れ子になっている。スキップ保護 plan は予想外にも数字が入力されなかったときに、0の割り算が平均値計算で生じるのを許さない。
- 結合：少なくとも2つの plan が交互に重なる。例えば、平均値問題を解決するために、入力累算 plan と入力数カウント plan は結合される。
- 適応：時々すでに開発されたお決まりの plan は、全く問題に必要とされるものではない。その状況で必要とされるように修正されなければならない。結局のところ、我々はそれをソフトウェアと呼ぶ。

plan 結合は効果的に行うのが非常に難しい機能である。(例えば、図1の虫つきプログラムでは、生徒は有効データ入力 plan を、見張り制御合計繰り返し plan と見張り制御カウンター繰り返し plan に、適切に結合できなかった。)初心者プログラマーが作るバグを研究すると、初心者が plan 結合を行うときに、いつも不適切に結合してしまうことが見受けられる。このような行為だけが驚くべきことではない。2つあるいはそれ以上の plan 合成は詳細に、多大な留意と注意を要求する。そして全ての微細な相互作用への予知能力を要求する。教育観点から、生徒は plan 結合問題から生じる問題に注意を怠ってはならない。

Plan 結合のための4つの方策を生徒に教えることは、弁明と技法をミクロレベルのプログラム部品で開発するのを明快にさせる。特に、全体目標解決のためにどのように下位目標をまとめるかを生徒が学ぶと、弁明は促進される。そして plan が実際のプログラミング言語構造用語で例示されると、技法は促進される。

プログラミング伝達の規則

実際のコードで plan を例示するとき、理解されるべき plan 方法にはとても融通性がある。異なる言語構成で plan を理解することが可能だし、plan の表現法は多種多様な指示を持ちうる。例えば、下図を考察せよ。正しい平均値を計算し、出力するものである。プログラミング指導者は考察状態ではこのプログラム作者を信用したがるな

い。なぜならこのプログラムがお粗末に書かれているからだ。一般に、0 か 1 の可変で変数を初期化することはあまりよい習慣ではない。この問題における目標と plan の分析は、生徒の変数初期化への心構えを明らかにしている。

```
Program Average
VAR count : INTEGER;
    sum, average, number : REAL;
BEGIN
    sum := -99999;
    count := -1;
    REPEAT
        writeln( 'Please input a number');
        read(number);
        sum := sum + 1;
    UNTIL(number = 99999);
    average := sum/count;
    writeln('the average is : ', average);
END.
```

読み込み目標、和算目標、入力数カウント目標を実現しようと、生徒はパスカルの繰り返し構造で、見張り制御合計繰り返し plan を実行した。しかしながら、繰り返し中に最初の読み込みで止まり値が入力される時、繰り返し構造は適切ではない。このような場合、繰り返し過程を実行すべきではない。パ斯卡ルでの全体構成には、見張り制御合計繰り返し PLAN による繰り返しが最も適している。繰り返し構造を使用すると、見張り値が間違っって合計を計算する恐れがある。バグによる欠点を埋めるために生徒は、プログラムの初期化段階で